MIT LIBRARIES DUPL 1
3 9080 00577702 1

MASSACHUSETTS
INSTITUTE of TECHNOLOGY
LIBRARIES

HD28
.M414
no. 3064-
89

DEWEY

MASS. INST. TECH.
OCT 4 1989

WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

What Good are Semistructured Objects? Adding Semiformal Structure to Hypertext

Thomas W. Malone*
Keh-Chiang Yu**
Jintae Lee*

SSM WP #3064-89-MS June, 1989

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

What Good are Semistructured Objects? Adding Semiformal Structure to Hypertext

Thomas W. Malone*

Keh-Chiang Yu**

Jintae Lee*

SSM WP #3064-89-MS June, 1989

*Massachusetts Institute of Technology, Sloan School of Management

**University of Kansas

OCT 4 1989

What Good Are Semistructured Objects? Adding Semiformal Structure to Hypertext

Thomas W. Malone, Keh-Chiang Yu, Jintae Lee

Massachusetts Institute of Technology

Conventional hypertext consists primarily of *nodes* containing unstructured text or graphics and *links* between these nodes (see reviews by Halasz, 1988; and Conklin, 1987). Thus most information in these systems is represented informally (e.g., as unstructured text or graphics) for processing by humans. The links provide only a small amount of formal structure for automatic processing, but the current interest in hypertext suggests how powerful even this small amount of structure can be.

At the other end of the spectrum are conventional knowledge-based systems where all the knowledge is represented formally (using structures such as frame inheritance networks and production rules) for automatic processing by the systems. These systems provide quite powerful tools for automatic reasoning, but encoding many kinds of knowledge using their rigid formal representations requires significant--and often completely infeasible--amounts of effort. It is only very recently that a few systems (e.g., Lai, Malone & Yu, 1988; Conklin & Begeman, 1988; Fikes, 1988; Harp, 1988) have begun to experiment with the semiformal middle ground between these two extremes.

In this paper, we are concerned with two questions: (1) What benefits (if any) can additional semantic structure provide for hypertext users? and (2) Are these benefits worth their cost in additional complexity for users? Our answers to these question are derived primarily from our early experience with the Object Lens system, a general tool for cooperative work and information management (Lai, Malone & Yu, 1988).

In answer to the first question, we will conclude that adding semantic structure to hypertext nodes can provide significant benefits for (a) summarizing the contents of objects and their relationships, and (b) automatically searching and manipulating collections of objects. In particular, the Object Lens system illustrates the surprising usefulness of customizable folders for summarizing collections of objects and rule-based agents for manipulating them.

We will also conclude that the additional complexity required for these benefits is reduced when (a) the structure in the objects is "soft" (i.e., the objects are *semi*-structured), (b) the additional features are incremental (i.e , users who don't use them don't need to know about them), and (c) the primitives for these additional features are at an appropriate level of abstraction.

We believe these properties characterize many successful *semiformal systems*, and the final section of the paper briefly defines such systems and summarizes principles for their design.

OVERVIEW OF OBJECT LENS

Object Lens is intended to be a general tool for cooperative work and information managment (cf., Goldstein & Bobrow, 1987; diSessa, 1985). One of its key goals is to let unsophisticated computer users create and modify their own applications without requiring the intervention of a skilled programmer. In designing such a general tool, a critical problem is defining a set of primitives at the "right" level, that is, not so low-level that they require significant effort to do anything useful, nor so high-level that they require significant modification whenever the users' needs change (see diSessa, 1986). In the case of Object Lens, we chose the following key primitives: (1) *semistructured objects*, (2) *customizable folders*, and (3) *semiautonomous agents*. Much more detail about the Object Lens system is given by (Lai, Malone, & Yu, 1988). Here we provide only a brief summary.

Semistructured objects. By defining and modifying templates for various semistructured objects, Object Lens users can represent information about people, tasks, products, messages, meetings, companies and many other kinds of information in a form that can be processed intelligently by both people and their computers. The system provides an interface to an object-oriented database in the sense that (1) each object includes a collection of fields and field values, (2) each object type has a set of actions that can be performed upon it, and (3) the objects are arranged in a hierarchy of increasingly specialized types with each object type "inheriting" fields, actions, and other properties from its "parents" (e.g., see Stefik & Bobrow, 1986). The objects are *semistructured* in the sense that users can fill in as much or as little information in different fields as they desire and the information in a field is not necessarily of any specific type (e g , it may be free text, a link to another object, or a combination

of text and links). Users can see and change individual objects, object type definitions, and object display formats through a particularly natural form of template-based user interface. These templates can also contain editable *embedded objects*.

Customizable folders. By collecting objects in customizable folders, users can easily create their own displays that summarize selected information from the objects in table or tree formats. For instance, they can select certain fields to be displayed in a *table* with each row representing an object in the folder and each column representing a field. They can also select fields from which the links between objects will be used to create a *tree* (or graph) display with each object represented as a node in the tree and each link in the selected field represented as a line between nodes.

Semiautonomous agents. By creating semiautonomous agents, users can specify rules for automatically processing the information in objects in different ways at different times. Agents can be *triggered* by events such as the arrival of new mail, the appearance of a new object in a specified folder, the arrival of a pre-specified time, or an explicit selection by the user. When an agent is triggered it applies a set of rules to a specified collection of objects. A rule contains a *description* of objects to which the rule will apply and an *action* to be performed on objects which satisfy the description. Descriptions are partially filled in templates specifying the values required in particular fields. *Embedded descriptions* can be used to specify properties of objects to which the original object is linked. Actions can be general actions such as retrieving, classifying, mailing, and deleting objects or object-specific actions such as loading files or adding events to a calendar.

We call these agents "semiautonomous" because they can take actions without the explicit attention of a human user, but they do not try to solve entire problems by themselves and human users and can easily see and change their processing rules.

Status of the system. The current version of Object Lens is intended as a "proof of concept" demonstration system; it is not engineered for regular use by large groups of people. This version is implemented in Interlisp-D on Xerox 1100 series workstations connected by an Ethernet. It makes heavy use of the object-oriented programming environment provided by Loops and the built-in text

editor, Tedit. As of this writing, the system has been used regularly by three people in our research group for almost one year, and several other people have used it regularly for shorter periods

An example: Representing the structure of arguments

To illustrate the benefits of adding semistructured objects to hypertext, we will describe an increasingly powerful series of sample applications in Objects Lens, each adding a class of benefits not present in the ones before. All these sample applications are designed to help people represent and manipulate the structure of arguments. For instance, a team of programmers designing a new computer system might use such applications to represent alternative design choices, the arguments for and against each one, and the rationale for the choices finally made. The application could thus help make the decision originally and store the decision rationale for later reference, such as when the program is being modified.

A number of systems have been built or proposed to provide this general kind of functionality [Conklin & Begeman, 1988, Stefik, Foster, Bobrow, Kahn, Lanning, & Suchman, 1987; Lowe, 1986; Smolensky et al, 1987; Lee, 1989). Our examples will be modeled primarily on the gIBIS system, since it is a recent and appealingly simple example of this class of systems.

Our primary purpose here is not to extend the state of the art in argumentation systems, but to show how easily many of these capabilities can be implemented in a general framework like ours that includes semistructured objects. In all but the very last examples, these capabilities can be implemented in Object Lens by knowledgeable end users in a matter of minutes or hours, with no special programming required. For example, our first implementation of most of these capabilities required less than two hours and used only the features of our system that are exposed to non-programming users.

Step 1: "Simple "hypertext

Figure 1 shows an example of how a "simple" hypertext system might support our argumentation application. This example is from the Object Lens system, but it uses only unstructured text nodes and links between nodes. Note that it is entirely up to you, the user of this system, to structure the information in the node and to put in the proper links. (In this example, and throughout the paper, we will use "you" to refer to users when that simplifies the explanation of system features.)

Step 2: Creating individual nodes

The next step in our series is to create semistructured object types that are appropriate for our application. For instance, the gIBIS system has three types of nodes: Issues, Positions (about an issue), and Arguments (which support or object to positions). Figure 2 shows an argument node that supports one position and objects to another one. The relationships between this node and the positions to which it refers are represented by links in the appropriate fields ("Supports" and "Objects to", respectively). When users click on a field name, a menu pops up containing suggested alternatives for that field. For instance, when you click on the Keyword field a list of suggested keywords appears and if they select one, it is automatically inserted in the field

To define the new node types (Issue, Position, and Argument), you specialize existing object types. For instance, you can create the new node types as specializations of Thing. To do this, you select the "Create Subtype" action on the type definition for Thing and then use the "Add Field" action on the new type to add the new fields (e.g., Supports, Objects to, and Author) that are present in this type, but not in Thing. In this case, the user also used the "Edit Field Names" action to rename the "Name" field (that was inherited from Thing) to be "Subject". To change the alternatives that are suggested by the pop-up alternatives menu, you can use the "Edit Alternatives" action. This action allows you to either directly edit a list of alternatives for a field, or to specify a folder whose contents at run-time will be displayed as the alternatives.

At this point, the system helps you create nodes with appropriate fields and alternatives, but it doesn't help you view the structure of the argument as a whole.

Step 3: Summarizing the contents and relationships in groups of objects

The next step is to add ways of displaying the overall structure of the argument using customizable folders. The original gIBIS system has a graphical browser that shows a tree format display of the nodes in the argument, and Figure 3(a) shows a similar display from Object Lens.

To create such a display in Object Lens, you can simply put all the nodes you wish to have displayed into a folder, select the "tree" display format for the folder, and select which links you want to have shown in the tree. For instance, in Figure 3(a), the user chose to display links from the Supports, Objects to, and Responds to fields. As the figure shows, links from each of these different fields is shown with a different kind of line (e.g., solid, dashed, etc.). (Note: It is more convenient to specify the display format for this folder if you first create an abstract type, say "gIBIS Node," whose specializations are Issue, Position, and Argument, and which contains all the fields contained in any of its children.)

It is also easy to display the same nodes in a table format display, selecting whichever fields you wish to be shown in the table (see Figure 3(b)). Figure 4 shows an additional feature of folders: the ability to select objects from another folder to be inserted into this one. For instance, the selection rule in Figure 4 selects all the counterarguments to a specific position.

At this point in the example, the system has most of the basic user interface functionality of the original gIBIS system. (Unlike the original gIBIS system, however, we have not implemented connections to a remote database server, nor have we hardened the system to the point where it can be used reliably by a large group of people.)

Step 4: Automatically selecting and manipulating objects

The last step in our example is to add intelligent agents to help search and modify the network of nodes. For instance, Figure 5 shows an agent like one you might use to notify you whenever people add arguments that support positions you have entered. This agent is triggered automatically when new objects are added to the folder containing the discussion of interest. Figure 6 shows the rule this agent uses to select the arguments that support a specific person's positions. This rule illustrates how embedded descriptions can be used to specify structural queries that depend on the link structure in the network as well as on the contents of intividual nodes. Figure 7 shows another (multiply embedded) rule that selects arguments that object to positions authored by people in a particular group (i.e., people who have a specific supervisor). This rule illustrates how queries can use information from throughout a user's knowledge base (such as knowledge about people and their supervisors).

Other applications

In addition to the argumentation example, we have used the Object Lens system to implement a number of other sample applications (e.g., see Lai, Malone & Yu, 1988). For instance, one of the first applications we implemented did local mail sorting like that done by the Information Lens system (e.g., see Malone, Grant, Turbak, Brobst, & Cohen, 1987). To do this, we simply defined various kinds of messages as object types, created agents that were triggered by the arrival of new mail, and specified rules to move new messages into folders. We have also recently developed demonstration applications to (a) support bug tracking in our research group, (b) help the administrative director of a research center track sponsors and projects, and (c) serve as a personal to-do list.

BENEFITS OF SEMISTRUCTURED OBJECTS

Now that we have seen examples of how end users can create powerful applications using semistructured objects and the other primitives provided by Object Lens, this section will analyze in more detail the benefits this additional structure provides.

More powerful tools for creating and acting upon individual objects

The benefits of structure for manipulating individual objects are mostly obvious: When the system knows the type of an object, it can suggest appropriate actions for different types of objects at different times. For instance, message objects that have just been composed can be Sent, and message objects received from elsewhere can be Answered or Forwarded. Semistructured objects can also help you create consistently structured nodes without having to retype or explicitly copy information from previous nodes. All Argument nodes in Object Lens, for example, contain the same fields in the same format, and when you create them you can get consistent guidance about what alternatives make sense in each field.

Powerful tools for summarizing the contents of objects and their relationships

The fact that the system "understands" something about the structure of nodes and the meaning of text or links in different fields allows the system to create much more useful summaries of the nodes in a folder than would otherwise be possible.

General display formats: Table and tree

Two display formats for folders, tables and tree, are useful for a wide range of object types and relationships. For instance, Figures 8(a) and 8(b) show two examples of folders containing task objects. The first folder summarizes the due dates, status, and responsible person for each task in a table format, the second folder shows a tree format display of the same objects. In this case, the display resembles a PERT chart that summarizes the prerequisite relationships among the tasks. It is important to realize that users can create displays like this for themselves. In this case, they would only need to define the task object with its appropriate fields (as in Figure 9), create folders with table and tree formats, and select the fields to be shown in the folders. Then any task instances the users create and place in the folders will be displayed appropriately.

Specialized display formats: Calendar

Most of our work so far has used two display formats for folders: tables and trees. These formats are applicable to a very wide range of object types and relationships. It is also possible, however, to create more specialized folder display formats for special kinds of objects. For instance, Figure 10 shows a specialized display format called "Calendar." The calendar format is used to diplay objects of type "Event." All Events (including specializations such as Meeting and Seminar), contain fields called "Date," "Start time," and "End time." When events are displayed in a calendar, these fields are used to locate the event in the proper day of a month (Figure 10(a)) and at the proper time within a day (Figure 10(b)). Users can select a month display like that in Figure 10(a); they can click on one day within the month display to bring up a day folder like that in Figure 10(b); and if they display an event within the day folder, the event itself is shown as in Figure 10(c).

Automatic agents for searching and manipulating networks

In addition to summarizing the contents of semistructured objects, the system can use their structure to perform even more powerful automatic actions such as searching and restructuring The Object Lens system uses rule-based agents to perform these automatic actions For example, Figure 11 shows an agent that maintains a folder of "Overdue Tasks." Every night at midnight, this agent is automatically triggered and searches the "*All Tasks" folder, a system-maintained folder that contains all task objects in the local workstation. When the agent finds tasks whose due date has passed, it moves them into the Overdue Tasks folder.

In general, agents can be used to retrieve objects and to automatically maintain folders according to specified criteria. But other kinds of actions can be defined as well. For example we are currently implementing several new types of actions to help provide facilities for automatic truth maintenance in argumentation networks like the one in our example above (see Lee, 1989). Imagine, for instance, that we created new types of nodes called "Assumptions" with links to the arguments that depend on them. Then, agents could look for assumptions whose truth value had changed, and update the truth values of the arguments and positions that depend on these assumptions

LESSONS

In designing and using the Object Lens system we have been struck with two kinds of lessons about semistructured objects. The first involves the nature of semiformal systems in general, the second involves how "soft" the object type definitions should be.

Semiformal systems

Elsewhere (Lai, Malone, & Yu, 1988), we defined a semiformal system as having three properties: (1) it represents and automatically processes certain information in formally specified ways; (2) it represents and make it easy for humans to process the same or other information in ways that are not formally specified; and (3) it allows the boundary between formal processing by computers and informal procesing by people to be easily changed. Semiformal systems are most useful when we understand enough about a situation to formalize in a computer system some, but not all, of the knowledge relevant to acting in that situation. We believe that this characterizes a very large number of the situations in which computers are likely to be used, both now and in the future.

By this definition, hypertext systems are clearly semiformal systems. In order to add more structure to hypertext systems without either losing the advantages of their semiformal nature or overwhelming their users with excessive complexity, our experience with the Object Lens system (and our observation of other systems as well) has led us to believe that the following principles are useful:

(1) The knowledge represented in the system is exposed to users in a way that is both *visible* and *changeable* (cf., Turbak, 1986). For example, users of the Object Lens system are able to easily see and change the object types and processing rules included in the system.

(2) There are successive *layers of exposure* of features in the system, and people can use features at one level without needing to know about deeper (and more complex) levels. For instance, some Object Lens users can create and edit instances of predefined object types and display them in predefined folders without ever knowing how to define new object types, folders, or agents.

(3) The primitives of the system are at an *appropriate level of abstraction*. It is, of course, difficult to know what the "right" level of abstraction really is, but the primitives in Object Lens appear to have desirable properties in this regard. As we saw in the examples, many useful applications can be created without excessive effort, and the customizations that users make seem close to the same level of abstraction they would use anyway in thinking about their task domain.

How "soft" should the structured objects be?

We anticipated in our original design of the Object Lens system that the structure in objects should be "soft" in the sense that fields could contain any combination of text or links to other objects. This softness allows the advantages of semiformal systems: unexpected situations can be easily accomodated informally without trying to "defeat" the system's restrictions. We were surprised, however, by a need to have objects that were--paradoxically--both "harder" and "softer" than we had originally intended.

"Softer" types. Our initial implementation of the Object Lens system did not allow people to change the type of an object after it had been created, and it required all changes in the properties of an object type to be made by editing the type definition itself. We now believe the system should include a "Change type" action on all objects to allow users to change their minds after originally creating an object. More importantly, we believe it should be easier to change object type definitions without leaving the context of editing an instance. For instance, there should probably be an option at the bottom of the pop-up alternatives menu to change the alternatives. It should also probably be possible to Add Fields to an instance of a type and to save modified instances as definitions of new types.

"Harder" types. We also believe that object types should be "harder" than we originally intended, in the sense that they should specify (but not rigidly enforce) expectations about the type of objects to be inserted in specific fields. Having the system know what types of objects are expected in each field allows the system to do useful things such as: (1) "soft" type checking (i.e., give you a warning when you try to put an unexpected type in a field), and (2) suggesting the type of object to be inserted in a

field (for example, the type of embedded description to be inserted in a rule like that shown in Figure 6).

Conclusions

We have seen in this paper how traditional hypertext and traditional knowledge-based systems lie near opposite ends of a continuum that measures the amount of formal structure and automatic processing embedded in the system. We have also seen how moving toward the structured end of this continuum can provide significant additional benefits to hypertext users. In particular, adding customizable folders provides powerful ways of summarizing collections of objects and adding rule-based agents provides powerful automatic tools for searching and manipulating collections of objects. Much of the usefulness of these systems appears to come from the fact that their semiformal nature allows users to have the best of both worlds: formal representation for automatic processing when appropriate, and informal representations for processing by people otherwise.

REFERENCES

Conklin, J. (1987). Hypertext: A survey and introduction. *IEEE Computer, 20*(9), 17-41.

Conklin, J. & Begeman, M. L. (1988, September 26-28). gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems, 6*, 303-331.

diSessa, A. (1985). A principled design for an integrated computational environment. *Human-Computer Interaction, 1*, 1-47.

diSessa, A. & Abelson, H. (1986, September). Boxer: A reconstructable computational medium. *Communications of the ACM, 29*, 859-868.

Fikes, R. (1988, August 23). Integrating hypertext and frame-based domain models. *AAAI 1988 Workshop on AI and Hypertext*, St. Paul, Minnesota.

Goldstein, I. & Bobrow, D. (1987). A layered approach to software design. In *Interactive Programming Environments*, D. Barstow, H. Shrobe, and E. Sandewall, Eds. McGraw-Hill, New York, pp. 387-413.

Halasz, F. G. (1988, July) Reflections on notecards: Seven Issues for the next generation of hypermedia systems. *Communications of the ACM, 3*, 836-852.

Harp, B. (1988, August 23). Position paper: Facilitating intelligent handling by imposing some structure on notes. *AAAI 1988 Workshop on AI and Hypertext*, St. Paul, Minnesota.

Lai, K. Y., Malone, T. W. & Yu, K. C. (1988, October). Object Lens: A 'spreadsheet' for cooperative work. *ACM Transactions on Office Information Systems, 6*, 332-353.

Lee, J. (1989). DLL (Decision Representation Language) and its support environment. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Working Paper #325

Lowe, D. (1984, February). Co-operative structuring of information: The representation of reasoning and debate. *International Jouranl of Man-Machine Studies, 23*(1), 112-125.

Malone, T. W., Grant, K. R., Turbak, F. A., Brobst, S. A. & Cohen, M. D. (1987). Intelligent information sharing systems. *Communications of the ACM, 30*, 390-402.

Smolensky, P., et. al. (1987). Constraint-based hypertext for argumentation. Department of Computer Science and Linguistics, University of Colorado Working Paper #CU-CS-358-87.

Stefik, M. & Bobrow, D. (1986). Object-oriented programming: Themes and variations. *AI Magazine, 6*(4), 40-62.

Stefik, M., Foster, G., Bobrow, D. G., Khan, K., Lanning, S. & Suchman, L. (1987). Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM, 30*, 32-47.

Turbak, F. A. (1986). *Grasp; a visible and manipulable model for procedural programs.* Unpublished M. S. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

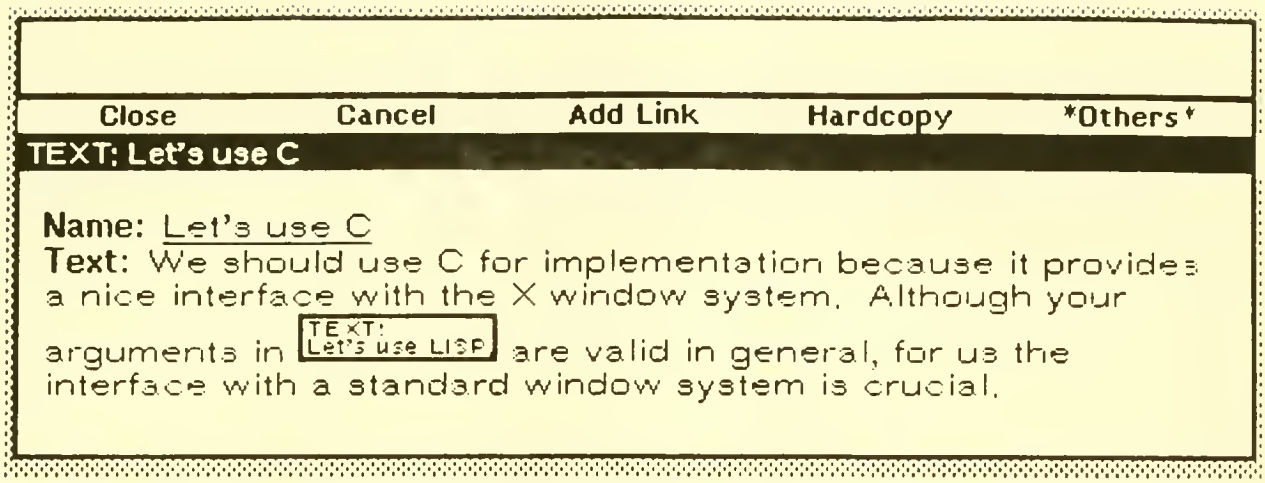


Figure 1.
A "simple" hypertext system uses unstructured text nodes and links between nodes.

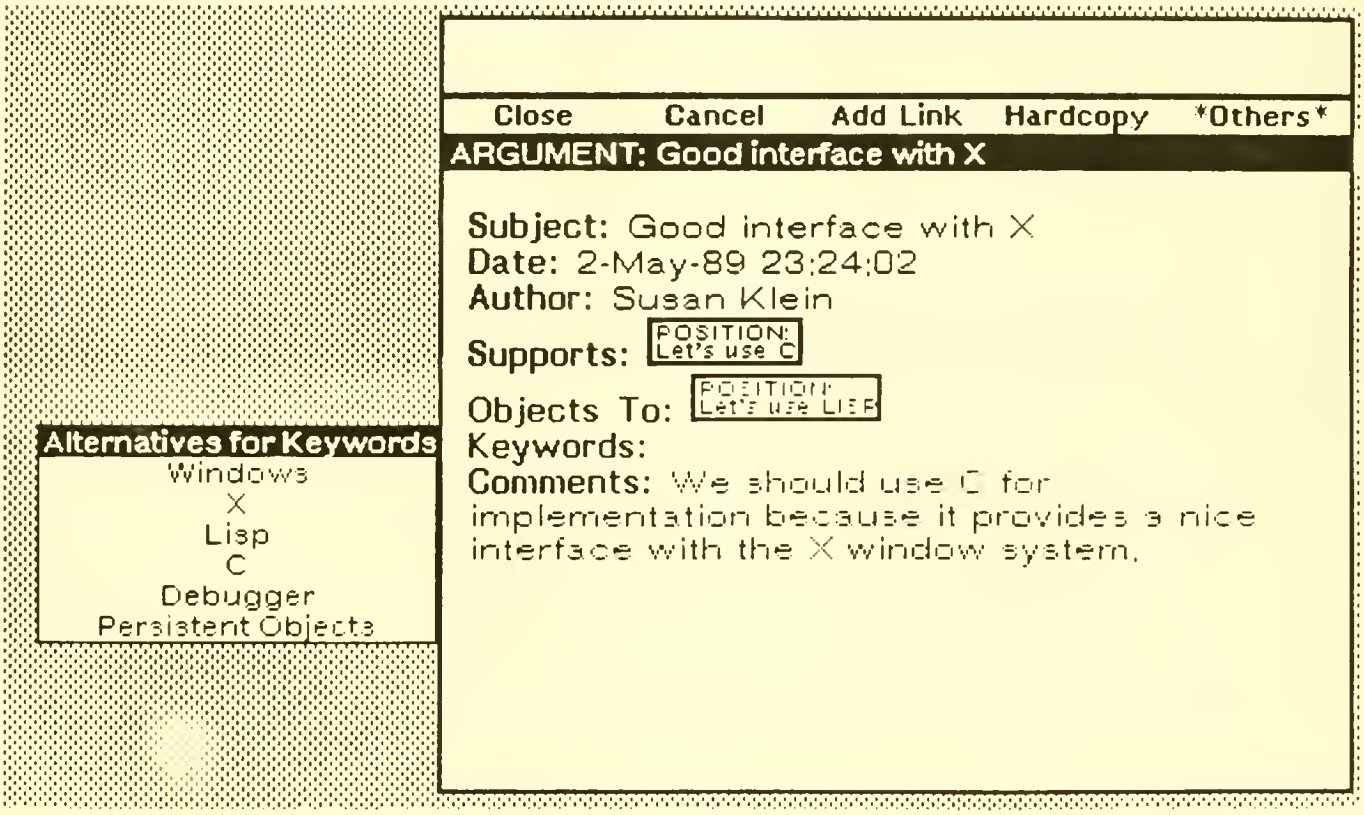
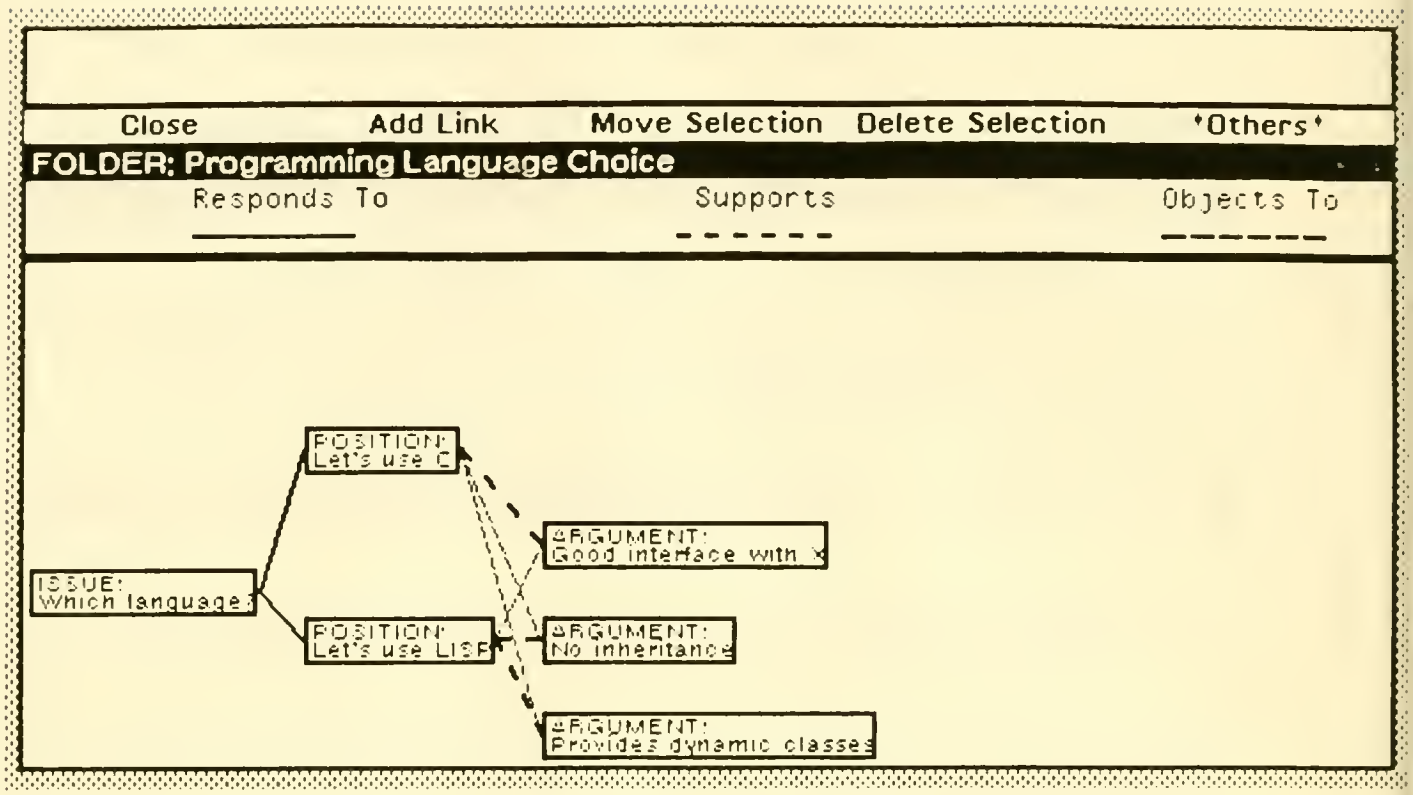
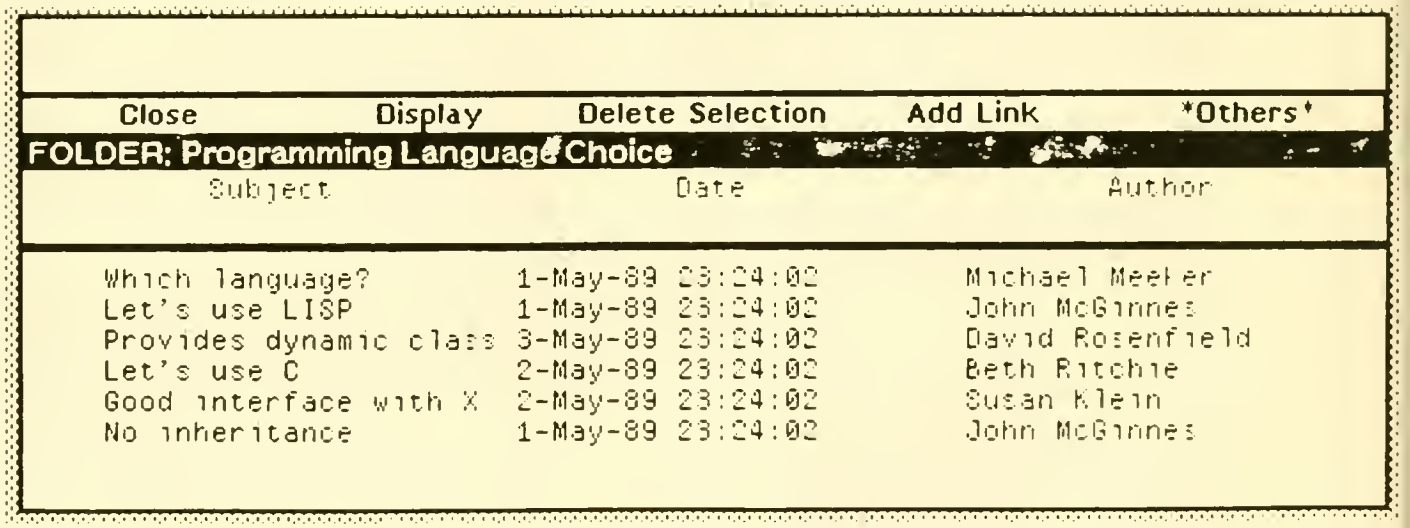


Figure 2.
Semistructured objects can be used to create nodes with as much or as little structure as desired



(a) Tree format display



(b) Table format display

Figure 3.
Customizable folders can summarize the contents of objects in a variety of formats easily tailorable by users

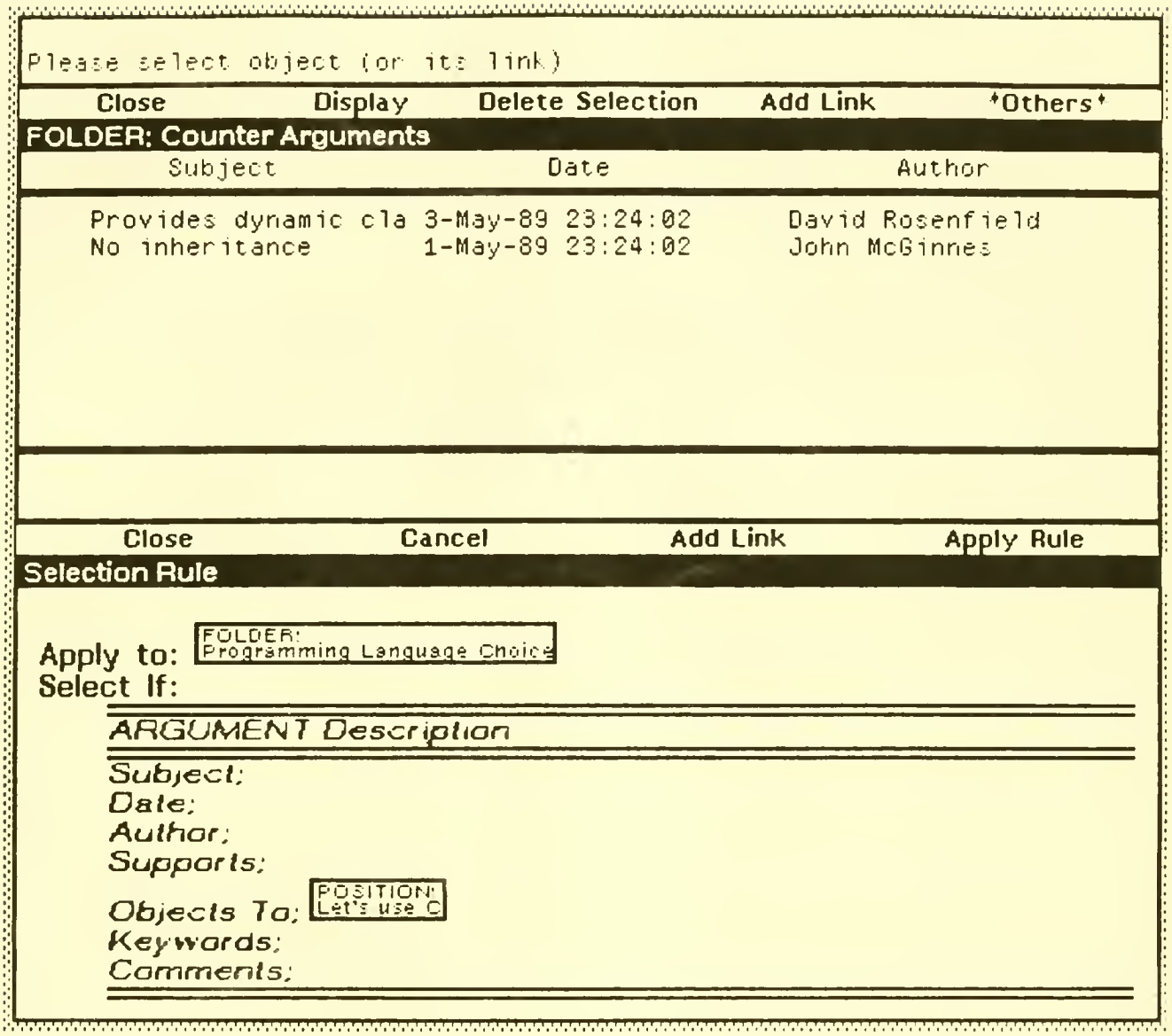


Figure 4.
Folders can use selection rules to select from another folder the objects they will include.

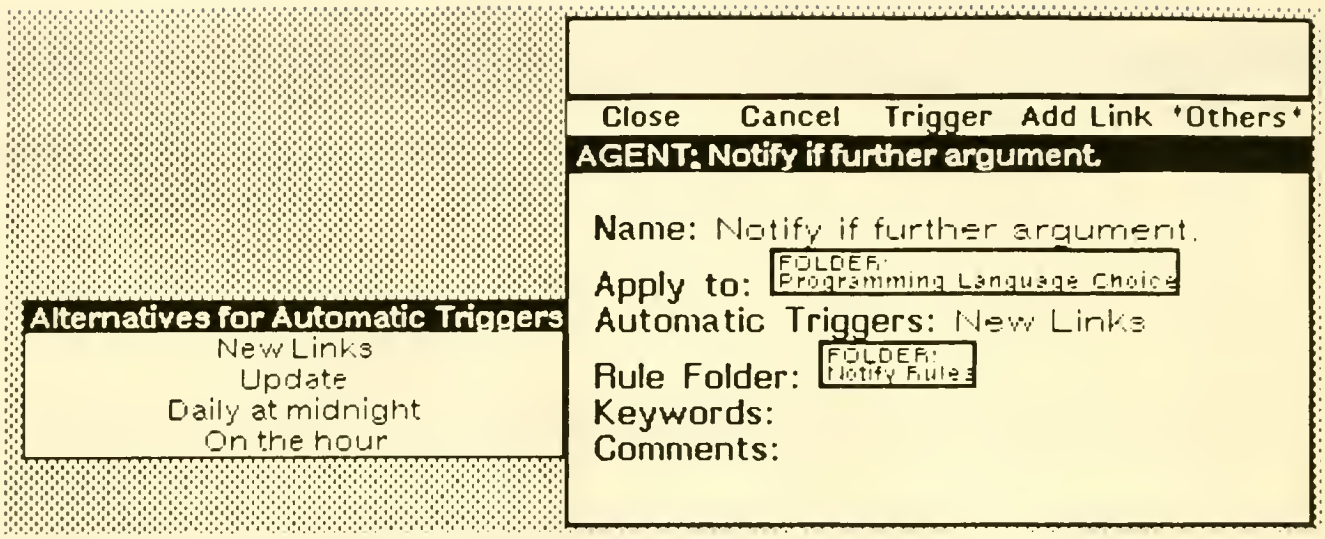


Figure 5.
Agents can either be triggered manually or automatically under various conditions. When triggered, they apply a set of rules to the objects in a folder.

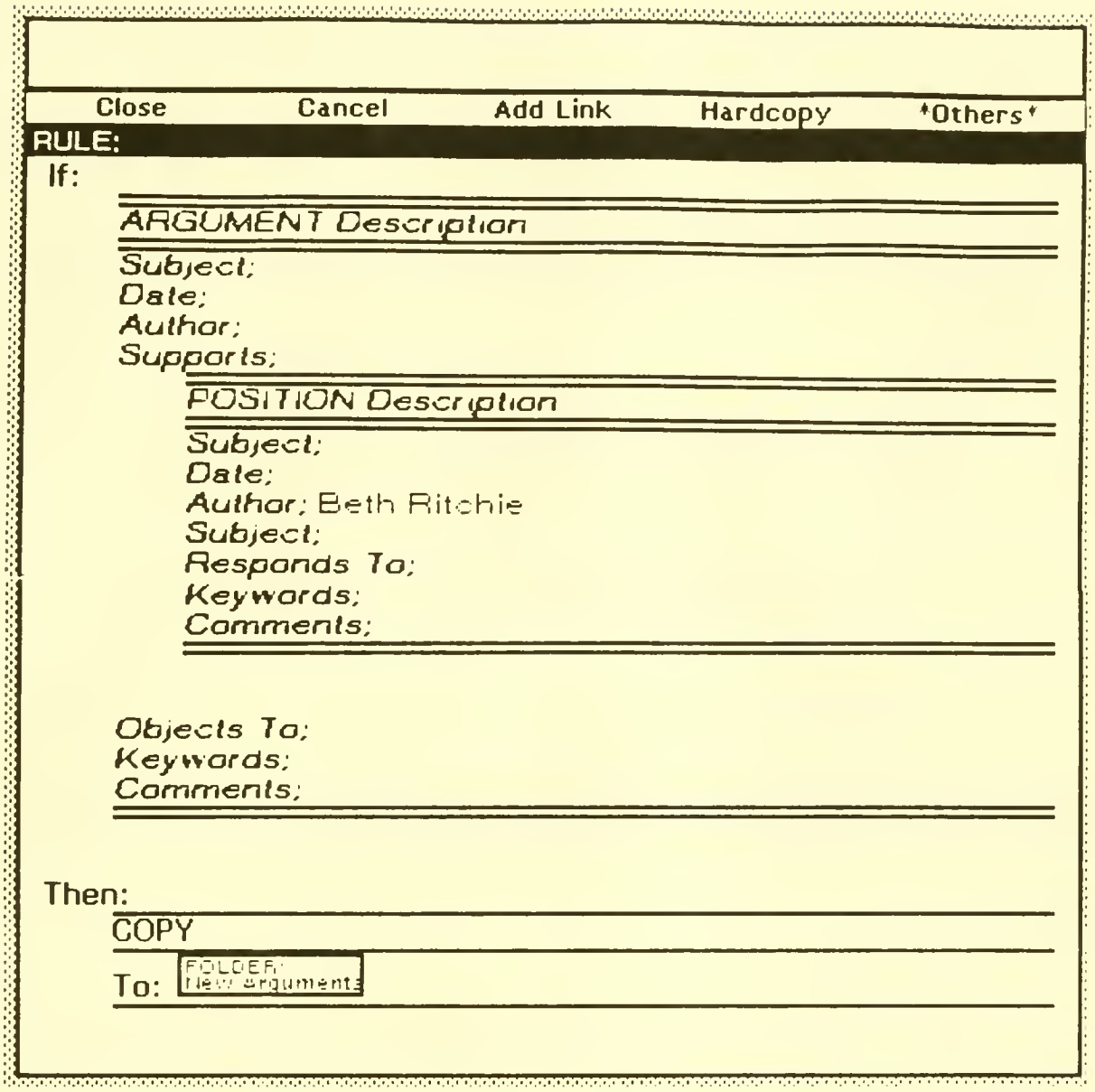


Figure 6.
Rules in agents describe the objects to which they will apply and the actions they will take. For example, this rule selects all arguments that support positions authored by a certain person.

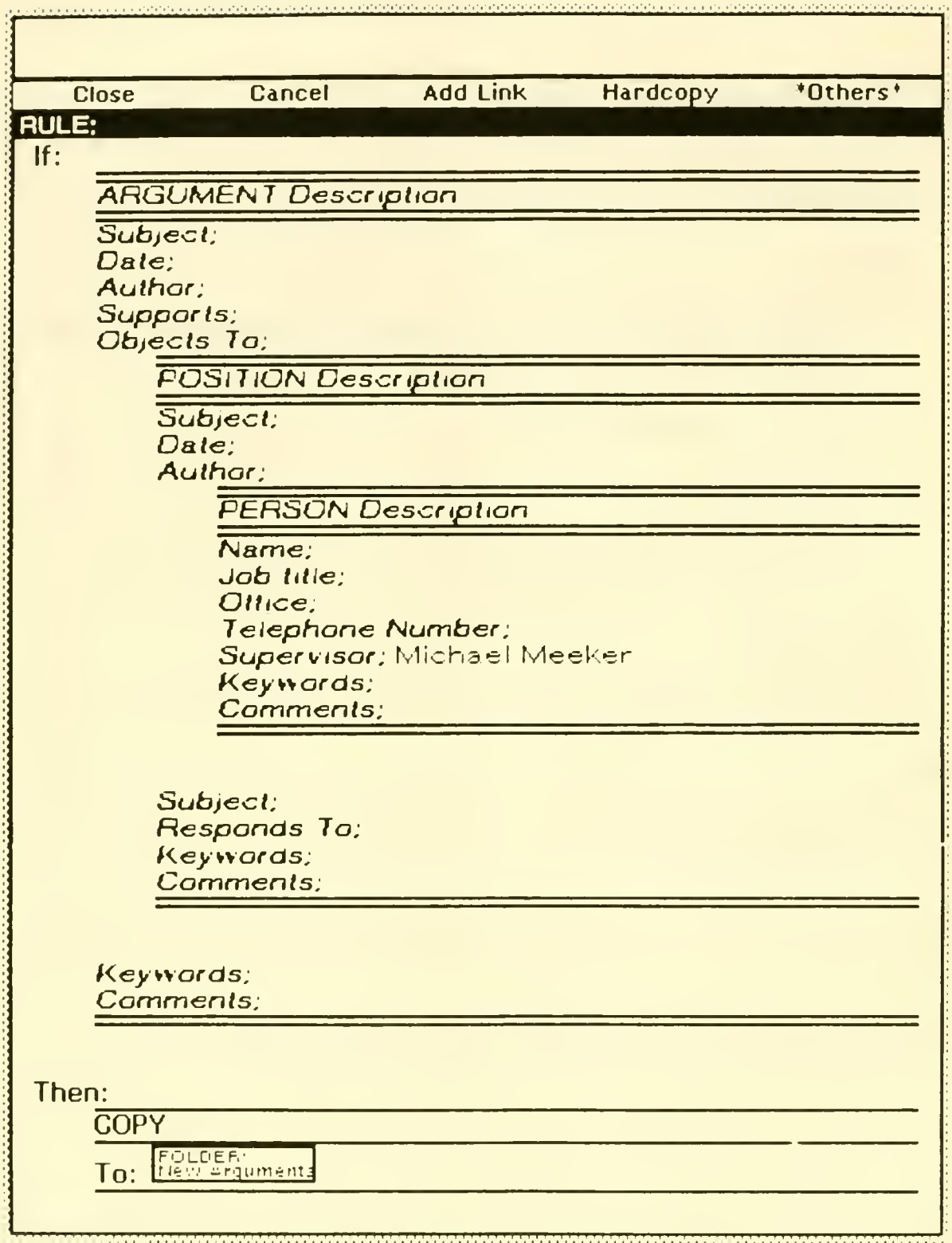


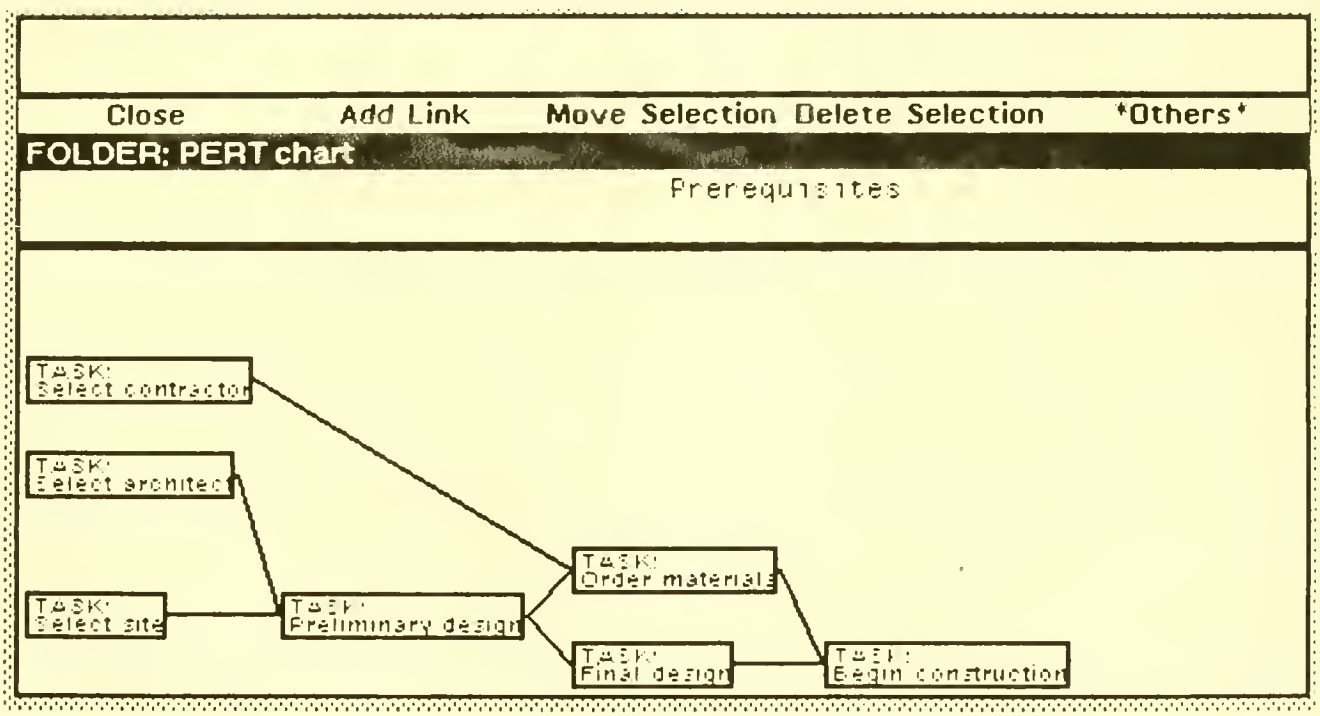
Figure 7.
Rules can use embedded descriptions to specify queries that depend on the link structure between nodes and on information from thoughout the knowledge base. For instance, this rule selects all arguments that object to positions entered by anyone with a particular supervisor (i e , anyone in that supervisor's group).

Close Display Delete Selection Add Link *Others*

FOLDER: Tasks

Name	Due Date	Status	Person Responsible
Select site	10-Jan-90	Done	Richard Berenson
▶ Select architect	20-Jan-90	*Overdue*	Elesse Brown
Select contractor	15-Jan-90	Done	Richard Berenson
Preliminary design	01-Mar-90	--	Architect
Order materials	30-Mar-90	--	Maya Bernstein
Final design	15-Jun-90	--	Architect
Begin construction	15-Jul-90	--	Maya Bernstein

(a) Table display format



(b) Tree display format

Figure 8.
These folders display task objects in a table format, highlighting due-dates and responsibilities, and in a tree format, highlighting the kind of task dependencies shown in a PERT chart.

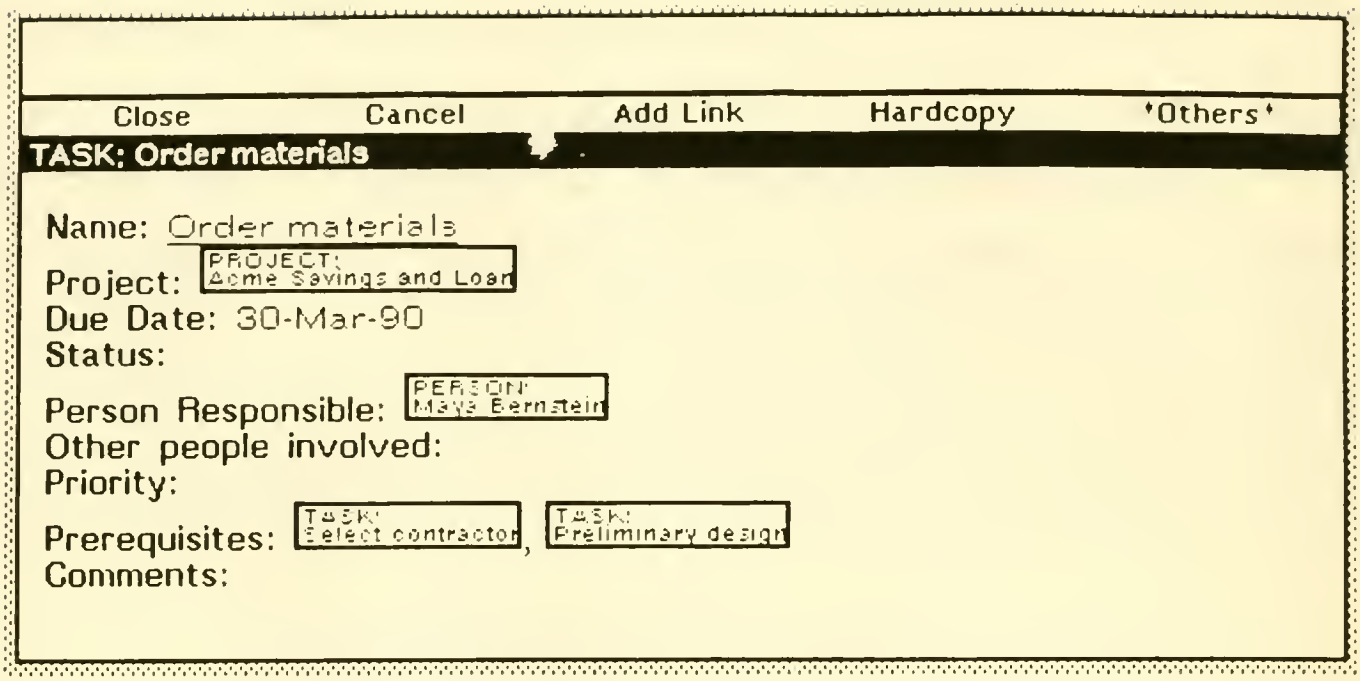


Figure 9.
Task objects like these can be displayed in folders like those shown in Figure 8.

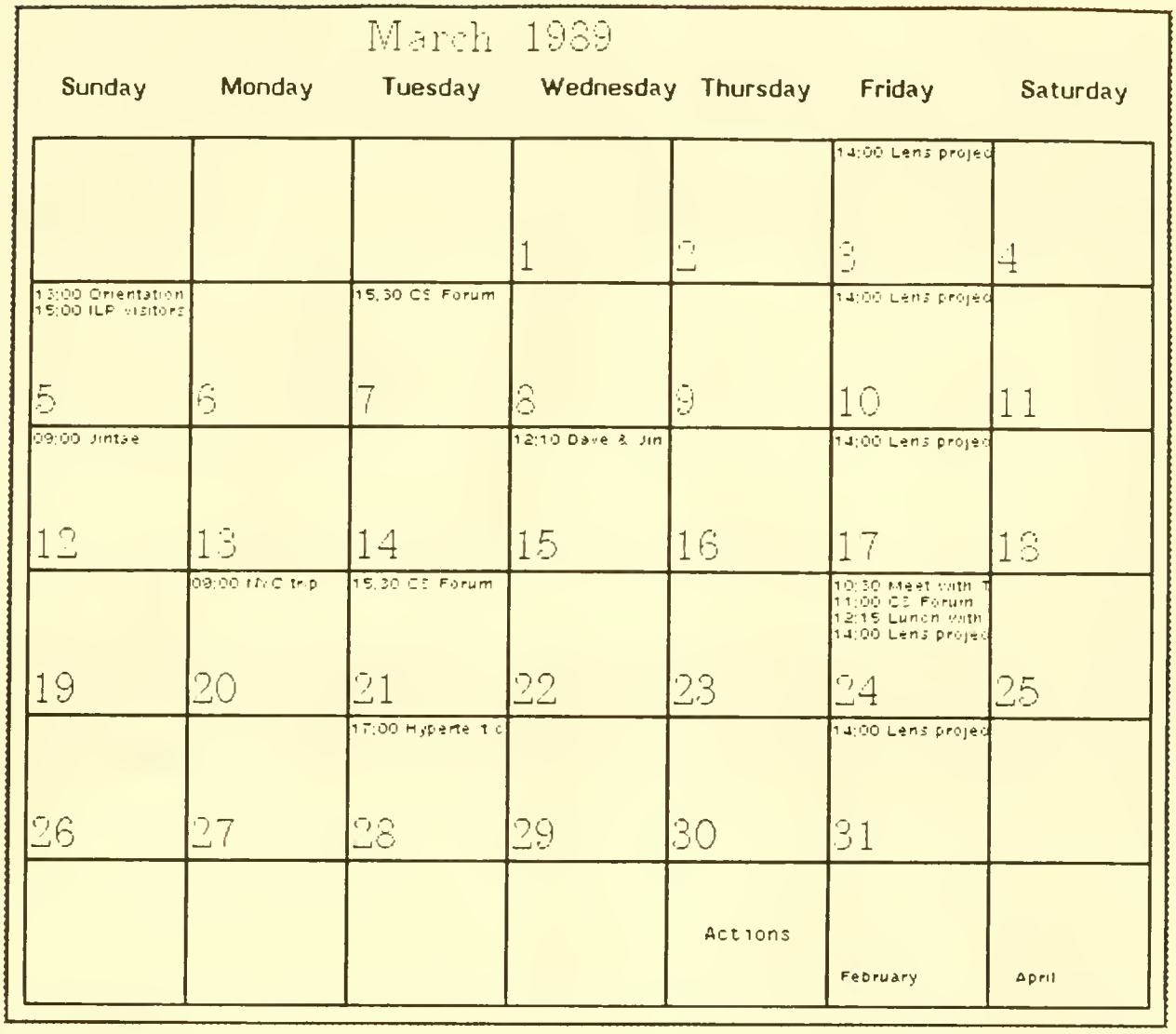


Figure 10(a)
Events in a folder can be displayed with a specialized "calendar" display format

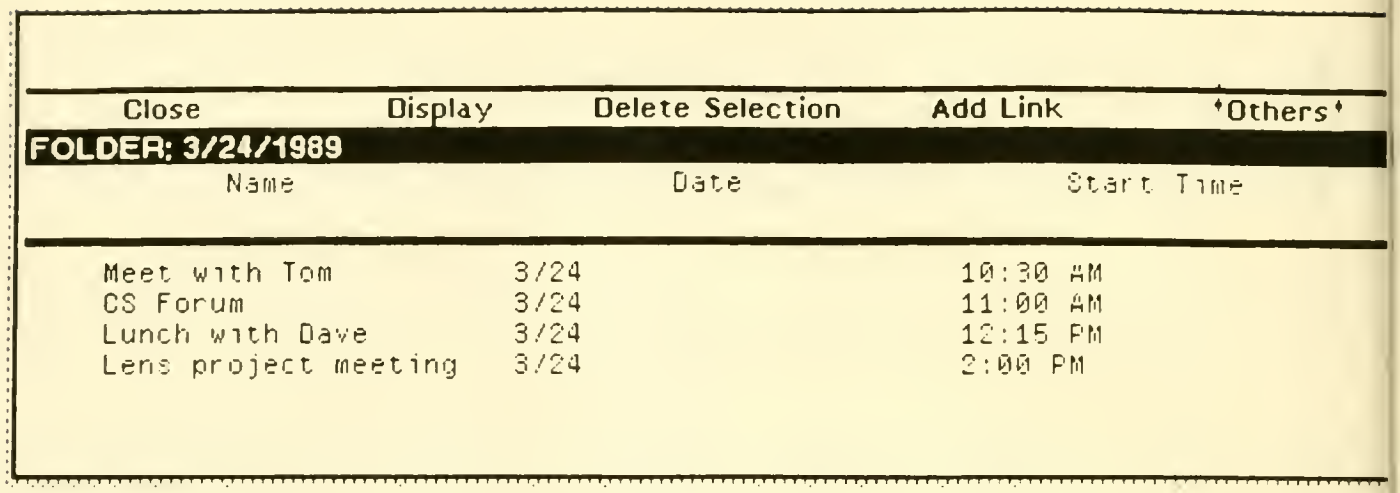


Figure 10 (b)
A day folder shows the events from one day of a calendar

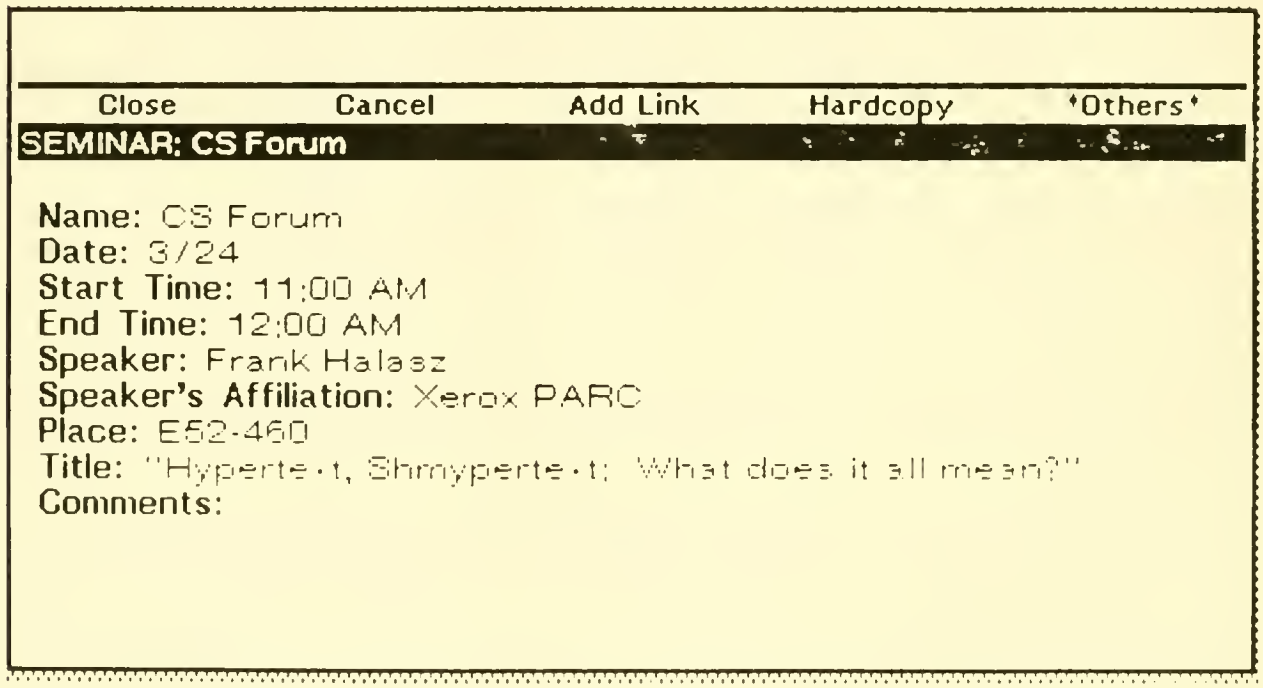


Figure 10 (c)
Seminars are among the specialized kinds of events that can be shown in calendars.

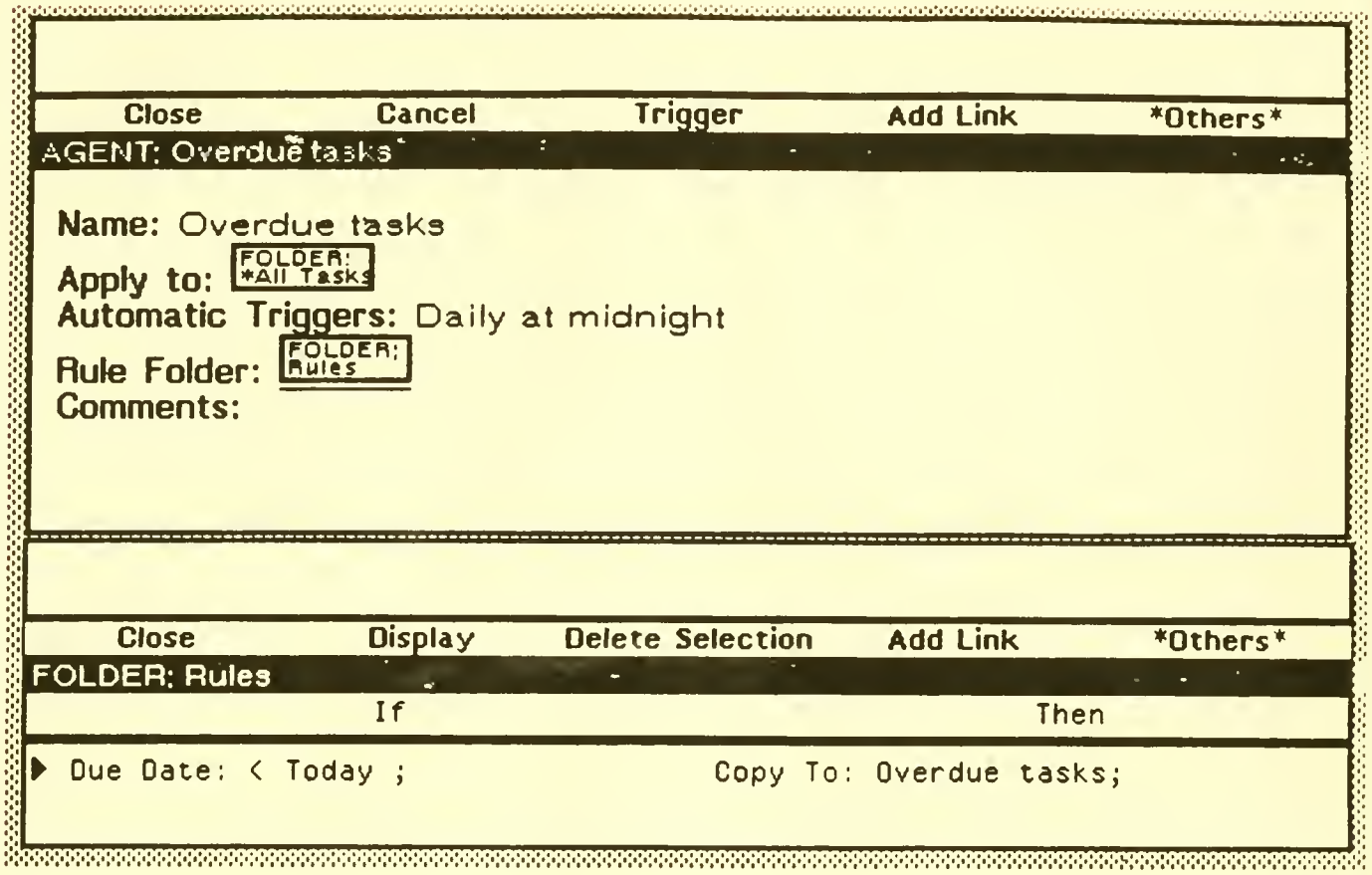


Figure 11.
This agent automatically moves overdue tasks to a special folder.

Date Due

JUL 19 1990

APR 15 1991

NOV 22 1995

SEP 30 1997

Lib-26-67

MIT LIBRARIES
3 9080 00577702 1